Proc. of the 1991 IEEE
Int. Conf. on Tools for AI
San Jose, CA-Nov. 1991

Selective Relaxation For Constraint Satisfaction Problems

E. C. Freuder **R. J. Wallace**

Computer Science Department
University of New Hampshire
Durham, NH 03824

Abstract

Two ways that arc consistency techniques are used to enhance efficiency of constraint satisfaction algorithms are: (i) as a preprocessing step that produces consistency between pairs of variables prior to global search, (ii) as part of a hybrid algorithm, to produce consistency ahead of the search process. A basic problem is to optimize the tradeoff between effort required to establish local consistency and that required for search. This report describes an approach to this problem termed *selective relaxation*. The idea is to perform consistency checking at places where it is likely to be effective, basing this judgement on local criteria. To this end, we introduce two forms of *bounded* relaxation, one in which consistency testing propagates for a limited *distance* from a point of change, and one in which it stops when the amount of change, or *response*, falls below threshold. Experiments show that these procedures can outperform well-known preprocessing or hybrid algorithms on many problems.

1 Introduction.

In the past decade, the study of constraint satisfaction problems has come to occupy an increasingly important place in the field of artificial intelligence. An enduring problem in this area is the difficulty of finding solutions to such problems. The classical backtracking algorithm, while much superior to ordinary depth-first search, is still exponential in the worst case. More importantly, it is common to find that an appalling amount of time is required to solve even small problems [Knuth 75]. To ameliorate this condition, methods have been developed that establish local consistency among subsets of variables in the problem, either prior to or during the search for a general solution. The best-known and possibly the most useful local consistency algorithms establish *arc consistency*, i.e., consistency between variable pairs that are subject to mutual constraints. Algorithms that do this are also called *relaxation* or constraint propagation algorithms, in that they relax the problem to a more 'stable' state by deleting inconsistent values from domains of successive constrained variables.

In some situations it may be more efficient to establish a limited degree of arc consistency than to carry out complete relaxation. This assumption underlies the use of directed rather than full arc consistency for preprocessing [Dechter and Meiri 89]. It is also reflected in hybrid algorithms that interleave limited forms of relaxation with search [Nadel 89]. The most notable of these is forward checking, in which consistency checking is limited to variables immediately constrained by the one last selected during search [Golumb and Baumert 65; Haralick and Elliott 80].

These examples show that the issue of tradeoffs in amount of local consistency checking versus checking during search is fundamental to considerations of overall efficiency. They also suggest that it would be useful to consider other ways to restrict relaxation, in the hope of finding more efficient methods.

In this report we describe a class of partial relaxation methods, termed *selective relaxation*, which suggests a new way of looking at this problem. In selective relaxation, consistency checking is carried out on variables that are likely to be affected by relaxation at each step of the procedure. Unlike forms of partial relaxation that follow a fixed order of processing, selective relaxation has the potential to be adapted to a given problem or a class of similar problems. Thus, if a problem is not amenable to relaxation at a given stage, a proper degree of selectivity may avoid useless tests. Successful relaxation, on the other hand, can entail further propagation. In either situation, it may be possible to adjust the selectivity of the process to reflect the conditions of the basic tradeoff more adequately.

The effectiveness of this approach depends on the criteria used to select variables for relaxation. A general strategy is to set bounds on the relaxation process. Two examples are described here: (i) constraint propagation is confined to a fixed distance (in terms of the constraint network) from the variable at which it began (*distance-bounded*

0-8186-2300-4/91 $01.00 © 1991 IEEE

relaxation), (ii) propagation along a path of the network stops when the amount of pruning from the domain of a variable falls below a certain threshold (*response-bounded* relaxation). ([Nadel 89] has also proposed a relaxation algorithm which is selective in that it only propagates from nodes with a single remaining value.)

Both forms of bounded relaxation have been tested with random problems either: (i) as part of the preprocessing step prior to backtrack search or forward checking, (ii) as an elaboration of the relaxation process in forward checking. Comparisons with the arc consistency algorithm, AC-3, in case (i) and with forward checking in case (ii) showed that selective relaxation can outperform these well-known algorithms in a variety of situations.

The next section, 2, reviews basic concepts and describes the AC-3 and forward checking algorithms. Section 3 describes the bounded relaxation procedures in some detail. Section 4 shows that bounded relaxation can overcome a basic limitation of forward checking that we call its "horizon". Section 5 describes the methods used to generate random constraint satisfaction problems and to test the algorithms, and Section 6 gives the results of testing bounded relaxation with these problems. Section 7 presents conclusions based on these studies.

2 Background: Constraint Satisfaction and Its Algorithms.

A constraint satisfaction problem (CSP) involves a set of n variables, v_i, each with a *domain* of values, d_i that it can assume. In addition, the problem is subject to some number of *binary constraints*, C_{ij}, each of which is a subset of the Cartesian product of two domains, d_i X d_j. A binary constraint specifies which pairs of values can be simultaneously assumed by the pair of variables. (Only binary constraints are considered here; higher-order CSPs can also be represented by binary CSPs [Rossi, Dhar and Petrie 89].) A CSP is associated with a *constraint graph*, whose nodes represent variables and arcs represent constraints.

A solution to a CSP is a combination of domain values, one from each variable, that are mutually consistent. There are two general classes of algorithms used to solve CSPs: backtracking and relaxation, or consistency, algorithms. Backtracking is a search procedure that examines the problem space in an efficient but exhaustive fashion to find one or all solutions. Most relaxation algorithms by themselves do not guarantee a solution. Instead, they achieve specific types of consistency among the problem variables. The most commonly studied form of consistency is called *arc consistency*; this is consistency between pairs of variables subject to binary constraints, so that the values in their domains are consistent with the constraint between them [Mackworth 77].

In the standard backtracking algorithm, variables are chosen in some order and each is instantiated with a value from its domain. After a variable has been instantiated, its value is checked against those of previously instantiated variables to see whether the new value is consistent with those already chosen. If an inconsistency is discovered, the new value is replaced by another and the latter is tested in the same manner; if there are no further values to test, the algorithm 'backs up' and tests another value in the domain of the last variable instantiated.

Initialize U to $\{v_i\}$, S to $\{\}$.
While U is not empty
  Select and remove v_i from U.
  Repeat until relaxation succeeds (no wipeout) or $d_i = \{\}$
    Set no wipeout
    Select and remove a value from d_i to instantiate v_i to v_i'.
    For each v_j such that v_i and v_j share a constraint
      relax v_j against v_i'
      if $d_j = \{\}$, set wipeout and break from for loop.
  If no wipeout add v_i' to S
  else
    if $i = 1$, report no solution and exit
    else
      put v_i in U with $d_i = d_i$ before instantiation and put v_l, the last variable instantiated, in U with $d_l = d_l$ - the last instantiation.

Figure 1. The Forward Checking algorithm (for one solution).

Forward checking also instantiates the variables in some order, but uses a different strategy for testing: rather than 'looking back', it 'looks ahead' after each instantiation, to the set of variables, v_j, that share constraints with v_i, the variable just given a value (Figure 1). The variables, v_j, are relaxed *against the instantiation*, to remove any values not supported by the value chosen for v_i. Since a variable that is instantiated has already been tested against prior instantiations, there is no need to look back, but this algorithm must still back up if there is no instantiation of v_i that supports at

least one value in each v_j. Forward checking is an example of a *hybrid* algorithm, one that combines backtracking and relaxation in its procedure.

Relaxation can be used to *preprocess* a CSP, i.e., to establish a level of consistency among some or all variables prior to search. This is done through a sequence of tests between pairs of constrained variables, v_i and v_j: we say that v_i is *relaxed against* v_j. Specifically, values in v_i are checked against the constraint between v_i and v_j to see if they are supported, i.e. are consistent with at least one value in the domain of v_j; unsupported values are deleted. An algorithm often used for preprocessing, which produces complete arc consistency, is AC-3 [Mackworth 77;.see Figure 2.] In this procedure, ordered pairs of constrained variables are first put in a list (L). Each pair, (v_i, v_j), is removed and v_i is relaxed against v_j. When values are deleted, pairs may have to be added to L to determine if these deletions lead to further deletions. (Although the arc consistency algorithm AC-4 [Mohr and Henderson 86] is better in the worst case, there is reason to believe that AC-3 is generally more efficient in practice.)

Initialize L to $\{(v_i, v_j)$ | there is a constraint between v_i and $v_j\}$.
While L is not empty
  Select and remove (v_i, v_j) from L.
  Relax v_i against v_j.
  If this relaxation removes any values from v_i, add to L any pairs (v_k, v_i), $k = j$, such that there is a constraint between v_k and v_i, and (v_k, v_i) is not already present in L.

Figure 2. The AC-3 algorithm.

---

3 Description of The New Algorithms.

When used in preprocessing, bounded relaxation is a refinement of the AC-3 algorithm. This entails testing each pair that is a candidate for inclusion in the list, to see if the variable to be relaxed falls within the stipulated bound: only then is the pair added. For distance-bounded relaxation, the bound associated with a single variable pair must be retained and updated. For this purpose, each pair of variables on the list is tagged with an integer representing the distance of the node to be relaxed from the "initiating node" (i.e., an original member of the list that was relaxed). Each pair added to the list is tagged with the value of the node just relaxed incremented by one. If a node is adjacent to one just relaxed but is already on the list, it is taged with the smaller distance. The resulting elaboration of AC-3 is shown in Figure 3.

Set distance bound, d.
Initialize L to $\{(0,v_i, v_j)$ | there is a constraint between v_i and $v_j\}$.
While L is not empty
  Select and remove (d_{ij}, v_i, v_j) from L.
  Relax v_i against v_j.
  If this relaxation removes any values from v_i,
    Set d_{next} to $d_{ij} + 1$.
    For each v_k, such that there is a constraint between v_k and v_i and $k = j$,
      If (d_{ki}, v_k, v_i) is not in L
        If d_{next} is less than or equal to d,
          Add (d_{next}, v_k, v_i) to L
      else
        Set d_{next} to the minimum of d_{next} and d_{ki} and replace(d_{ki}, v_k, v_i) with (d_{next}, v_k, v_i)

Figure 3. AC-3 with distance bound.

In response-bounded relaxation, if the degree of relaxation of v_i is greater than or equal to response bound, r, then pairs (v_k, v_i) are added to the list of pairs provided they meet the other restrictions described in Figure 2. The only elaborations of pseudocode required in Figure 2 are, therefore,

"Set response bound, r."

at the beginning and

"and the proportion of values deleted is at least r"

at the end.

Note that in the limiting cases, when the distance bound is zero or the response bound is 100 (percent), no pairs are added to the list.

Both types of bound can also be used to enhance relaxation in the forward checking procedure. In the present implementations, any relaxation that is within the bound is followed by relaxation of adjacent nodes within the set of future nodes, i.e., those not yet instantiated. After each instance of domain restriction, further propagation is preceded by a test, either of the present distance (number of steps) from the instantiated node or of the amount of restriction. If the bound has been reached, in the case of a distance bound, or if the bound has not been met, in the case of a response bound,

there is no further propagation from that point. In this case the limiting bound is 1 for distance bounds and 100 for response bounds, both of which give ordinary forward checking. In terms of the pseudocode of Figure 1, the statement under "For . . ." becomes:

"if d_j = { }, set wipeout and break from for loop
else if relaxation of v_j is within the bound
  Set U_B to the singleton $\{v_j\}$.
  While U_B is not empty
    Select v_k from U_B.
    Select $\{v_l\}$, the variables adjacent to v_k and
      for each v_l, if relaxation is within the bound,
      add v_l to U_B."

4 Forward Checking and 'Horizons'.

As described in Section 2, the relaxation step in forward checking involves testing the domains of the variables adjacent to the last one instantiated. This is a very conservative relaxation strategy, which may fail to detect values in the domains of uninstantiated variables that must later be deleted. We can say that forward checking operates with a very limited 'horizon' in terms of inconsistencies among the remaining variables in the constraint network [*cf.* Berliner 73]. Of course, there are cases in which a greater amount of relaxation is not worth doing. This was shown by [Haralick and Elliott 80], who found that more extensive consistency checking diminished the efficiency of the hybrid process for the problems they studied. But, if one were able to carry out extra checking selectively, the potential benefits of more extensive checking might become available. This is precisely what selective relaxation is designed to do.

The potential importance of the horizon effect in limiting the efficiency of forward checking is shown here by constructed examples. These CSPs have five variables (0, 1, ..., 4), each with a domain of four values (0, 1, 2, 3). The constraint graph is a simple cycle. A section of the domain support structure for one problem, i.e., the pattern of pairings of values in adjacent domains, is shown in Figure 4. Overall, this structure is a forest, with each tree rooted at a value of the variable where search begins, in this case variable 0. (For this problem, another tree, associated with value 3 of variable 0, is similar to the one shown here, but it 'branches out' in the opposite direction. The two trees associated with values 1 and 2 are also mirror images.)

The important point to note is that, if search begins with variable 0, marked asymmetries are encountered in the pattern of associations for each value of this variable, depending on the direction of search. In particular, the fact that there are no solutions that involve values 0 or 3

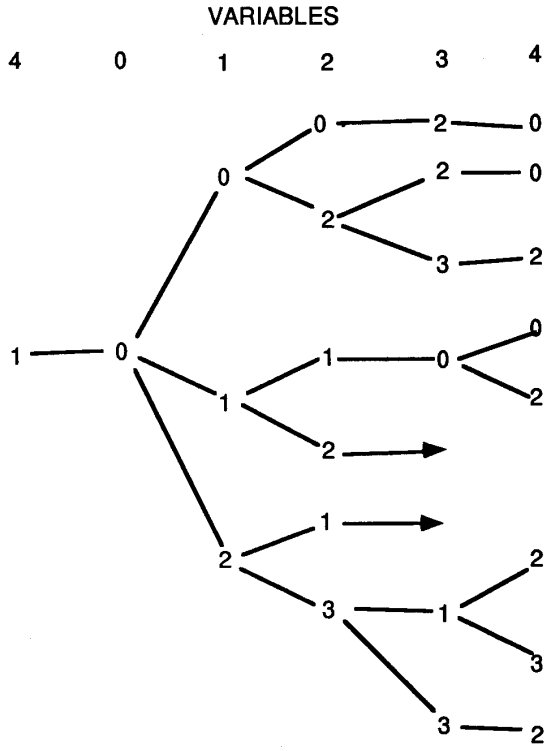

Figure 4. Domain support structure rooted at value 0 of variable 0 in a CSP that shows a horizon effect (CSP 1 in Table 1). Arrows indicate duplicate subtrees for values 1 and 2 of variable 2 in the diagram.

of variable 0 is determined more efficiently when relaxation proceeds in the direction 4, 3, 2, 1 in the first case and 1, 2, 3, 4 in the other. Forward checking will, therefore, overlook this feature at first in one of these cases. Bounded relaxation, on the other hand, can detect this feature almost immediately in both cases, saving considerable work for later stages of relaxation. This is reflected in the results of Table 1. (Since the domains are the same size to begin with, the response bounds used are absolute values. Note that here the number of pair checks is separated into those occurring during forward checking ["FC"] and those occurring during the extra bounded relaxation ["BD"].) Obviously, this effect is monotonic in the number of affected domain values (cf. CSP 2 in Table 1, for which the asymmetries for values 0 and 3 are in the same direction), and an increasing effect

should also be found, based on the length of the cycle.

Table 1
CSPs Constructed to Show Benefits of Relaxing Selectively beyond Forward Checking's Horizons

			Total Pair Checks for All Solutions					
			Fwd Chk Alone	Distance Bnd (max)			Response Bnd (min)	
				2	3	4	1	2	3
CSP 1	FC	111	50	46	43	44	50	54
	BD	---	37	63	83	69	42	26
	TOT	111	87	109	126	113	92	80
CSP 2	FC	152	54	50	46	48	54	58
	BD	---	44	80	104	76	42	26
	TOT	152	98	130	150	124	96	84

The basis for the gain in efficiency shown in Table 1 can be deduced from the traces of Figure 5, where the current pair check count is printed beside the next domain value selected. If the counts associated with values of variable 0 (far left of each trace) are compared, it is clear that the main difference in favor of bounded relaxation is found when this variable is instantianted with value 0, giving 19 pair checks in one case and 53 in the other.

5 Tests with Random Problems. Methods.

The above results show that there are problems in which bounded relaxation can lead to greater efficiency. Data pertinent to the question of the frequency of such effects was obtained with random CSPs. In the major problem set each CSP had 12 variables and a maximum domain size of 12. A limit of 1000 solutions was chosen to balance the requirements of representativeness and tractability. The number of binary constraints, the specific variable pairs subject to constraint, the size of each domain, the number of acceptable pairs in each constraint, and the specific value pairs in that constraint were chosen in this order using random selection of values at each step. This procedure allows every value of each parameter to be chosen with an equal probability. It also produces problems that are heterogeneous in the satisfiability, domain size, and degree of node in the constraint graph. However, owing to the apparent scarcity of problems with a large number of constraints that also have solutions, the number of constraints in these problems was limited to the range of 11 to 29, inclusive. Twenty such problems were obtained, and the results reported here have been corroborated with examples from a set of 40 six-variable problems generated with the same procedure.

Forward Checking	Distance-Bounded Relaxation
0 pc=0	0 pc=0
0 pc=8	1 pc=19
0 pc=12	2 pc=35
2 pc=16	1 pc=37
2 pc=17	0 pc=42
2 pc=21	0 pc=43
3 pc=22	3 pc=43
1 pc=23	2 pc=49
1 pc=27	2 pc=65
0 pc=31	1 pc=67
2 pc=32	0 pc=72
2 pc=36	0 pc=73
3 pc=37	3 pc=73
2 pc=38	3 pc=79
1 pc=42	
0 pc=46	
3 pc=47	
1 pc=51	
3 pc=52	
1 pc=53	
2 pc=61	
1 pc=65	
0 pc=69	
0 pc=70	
3 pc=70	
1 pc=74	
3 pc=75	
2 pc=76	
2 pc=84	
1 pc=88	
0 pc=92	
0 pc=93	
3 pc=93	
1 pc=97	
3 pc=98	
3 pc=99	
3 pc=107	

Figure 5. Full traces of forward checking and distance-bounded relaxation for CSP 1 of Table 1, with a distance bound of 2.

In each case the response bound values (i.e., the smallest fraction of domain restriction that led to further relaxation) were 0, .25, .50, .67, .75 and 1.00, the latter yielding either no additions in preprocessing or ordinary

forward checking in the hybrid context. For preprocessing, the distance bounds varied between 0 (no additions) and 4; for the hybrid case the bounds varied between 1 (equal to forward checking) and 5.

Care was taken to incorporate the effects of variable ordering, either of the node-pair list in AC-3 or the search order in forward checking. In the former case, (i) the list was treated as a queue, with pairs added and removed in FIFO order, or, (ii) the list was ordered by increasing size of the domain to be relaxed against. Previous studies have shown that order (ii) is a particularly good heuristic for enhancing the efficiency of AC-3 [Wallace and Freuder 91]. For forward checking, several fixed search orders, in addition to an ordering based on the lexical order of the variable names, were examined on the basis of earlier findings of enhanced efficiency [Haralick and Elliott 80; Dechter and Meiri 89]: increasing domain size, decreasing degree of nodes in the constraint graph, and reversed minimum width. (A minimum width ordering is obtained by removing a node of minimum degree, together with its adjacent arcs, placing it at the head of the list, and repeating the process until all nodes have been removed [Freuder 82].)

The measure of efficiency of performance was the total number of value pairs checked for consistency. Differences in this measure related to variable ordering or to bound value were evaluated statistically. The existence of differences due to either factor was tested with the analysis of variance (ANOVA). If the ANOVA showed a statistically significant effect of bound value, and if the mean performance favored bounded relaxation over forward checking or AC-3, paired comparison t tests were used to compare the best bound values with the reference algorithm. Up to three tests were performed in any test series (i.e., for any combination of solutions sought (one or all), bound type (distance or response) and ordering), using the three best bound values overall (0.50, 0.67 and 0.75 for response bounds and 1, 2 and 3 for distance bounds).

6 Tests with Random Problems. Results.

Results for forward checking with additional response or distance bounded relaxation are shown in Figures 6-8. Figures 6 and 8 show results for the all-solutions problem, Figure 7 for the one-solution with response bounds. Results for the one-solution problem with distance bound were similar to the all-solution case with this bound.

For all forward checking series, the ANOVAs showed a statistically significant effect of bound value. For random lists, mean performance was markedly improved by selective relaxation beyond forward checking's horizon. This was most obvious for response-bounded relaxation, with differences in favor of this procedure observed for all bound values. (These differences were statistically significant for all three comparisons for the all-solutions problem, with probability, p, less than 0.05, but not for the one-solution case, evidently owing to interproblem variability.) For distance-bounded relaxation, a statistically significant difference was found for the bound value 2 in the all-solutions case ($p < 0.05$).

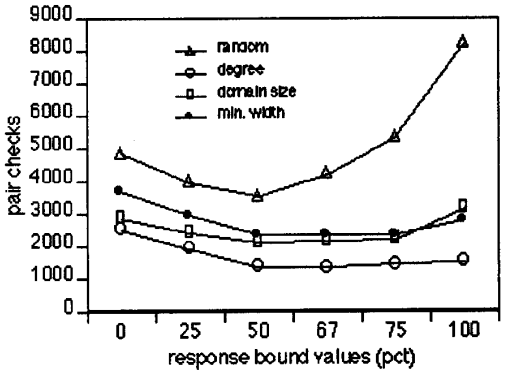

Figure 6. Mean Performance for the all-solutions problem with random CSPs as a function of response bound. A bound value of 100 gives forward checking.

As expected, fixed order heuristics improved performance in comparison with random ordering. In these cases, the effect of selective relaxation was reduced. For response-bounded relaxation it was still present for all orderings, although it could only be detected for the all-solutions case. Here, all three comparisons were statistically significant ($p < 0.05$) for ordering by domain size, and one comparison (involving the bound value 0.67) for ordering by degree. In contrast, mean performance with distance-bounded relaxation was always inferior to that of forward checking with these orderings. However, the variation in performance, as indicated by the standard deviation, was always greater for forward checking than for the best response or distance bound values, regardless of ordering.

In all test series with forward checking, there was a reduction in the number of dead ends encountered, when selective relaxation was employed. This effect was observed for most problems with each type of ordering. It was tested statistically by comparing the number of dead ends that forward checking encountered with corresponding values for the response bound value of .75

and the distance bound value of 2, respectively, since other bounds yielded even lower numbers of dead ends. This comparison was statistically significant in all cases, i.e., for both types of bounds, both the one- and all-solution problems, and for all orderings.

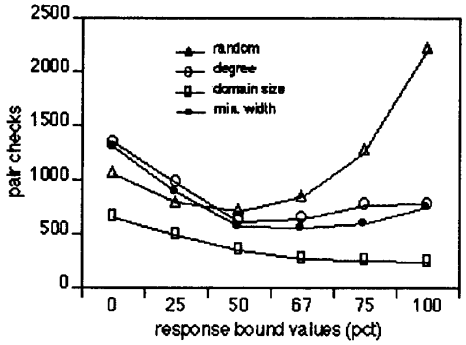

Figure 7. Mean Performance for obtaining one solution for random CSPs as a function of response bound. A bound value of 100 gives forward checking.

For the preprocessing series, comparisons with AC-3 also showed improvements over the basic algorithm for the queue ordering (Figure 9: pair checks). Unfortunately, the largest amounts of savings for this measure (for values of 0 and 1 for the distance bound and .75 and 1.00 for the response bound) were associated with lessened improvement in the efficiency of forward checking in comparison to its performance after full arc consistency had been achieved. (Here, a difference of one percent in the mean performance of forward checking is equal to about 100 value pair checks.) The best overall results were obtained with low response bound values, in which full relaxation was achieved without putting every variable pair back on the list; in this case, fewer checks were made during preprocessing, while performance during subsequent search was unaffected.

When the more efficient ordering heuristic was used with AC-3, bounded relaxation usually had a proportionally smaller effect on performance (about one-half), although there was only a one percent decline in improvement in forward checking even with extreme bound values (i.e., low distance or high response values). Given the considerable enhancement of relaxation efficiency effected by good orderings for these problems [Wallace and Freuder 91], it is not surprising that the improvement due to bounded relaxation was less pronounced. It is encouraging, however, that, in terms of the overall tradeoff, there was still a slight advantage.

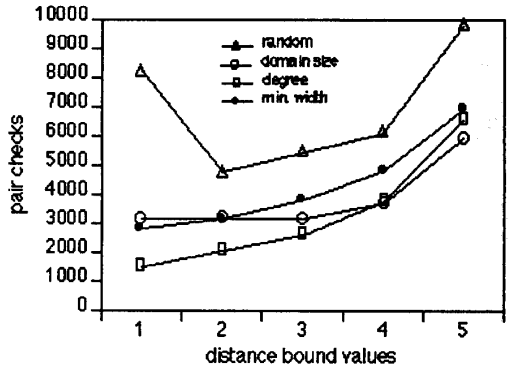

Figure 8. Mean Performance for the all-solutions problem for random CSPs as a function of distance bound. A bound value of 1 gives forward checking.

7 Conclusions.

These results show that bounded relaxation methods can improve the efficiency of forward checking on many problems. Some improvement was also found when bounded relaxation was used in connection with preprocessing, for a commonly used list ordering. However, the marked effect of good ordering on the performance of AC-3 apparently left little for these procedures to do, although it is possible that effects might be seen in this case with larger problems.

Although the features of these random CSPs distinguish them from the problems used by other investigators, e.g. [Haralick and Elliott 80], our sampling procedure ensured a greater variety of problems with more variation in the basic features. Methods that work well on them are, therefore, more likely to be of practical importance for this reason alone. Clearly, however, the problem of classifying CSPs in relation to the effectiveness of these procedures is an important area for future investigation.

In addition to differences in mean performance, the finding that bounded relaxation produced a consistent decrease in the number of dead ends encountered is a promising result, because the need to backtrack from dead ends is an important cause of poor performance during search [Dechter and Meiri 89; Freuder 82; Mackworth and Freuder 85]. It is also significant that variation in efficiency across problems was reduced by bounded relaxation; this suggests that these procedures can avoid

situations that result in very poor performance, where ordinary forward checking does not.

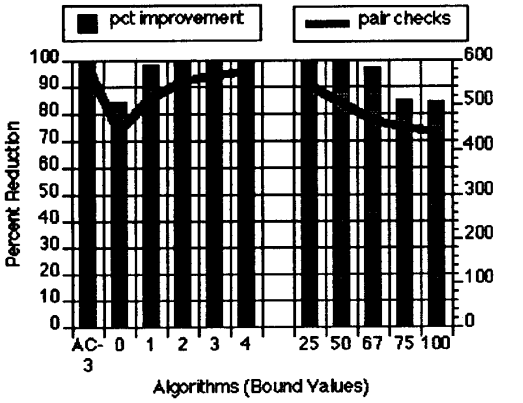

Figure 8. Number of pair checks performed during relaxation and proportional improvement in forward checking due to arc consistency, for bounded relaxation procedures in comparison to AC-3.

On the whole, response-bounded relaxation seems to be more effective than distance-bounded relaxation. This suggests that more precise assessment, at each step in constraint propagation, is necessary to achieve the best performance. (This is corroborated by subsidiary tests on these problems with response-bounded relaxation, using as bounds the absolute numbers of values deleted; this approach was less efficient than the use of proportional bounds in the experiments reported here.)

As noted earlier, bounded relaxation is only one class of selective relaxation strategies. Another form of selective relaxation, for which we have preliminary results, is based on calculations of expected domain sizes. Tuning of bounds can also be done in terms of feedback; this would give the procedure learning capacities and might allow it to adjust itself to the characteristics of a particular problem or problem domain, to insure efficient performance. These are possibilities that can be explored in the future, building on the general concept of selective relaxation, whose usefulness is supported by the results presented here.

Acknowledgements.

This work was supported by the National Science Foundation under Grant No. IRI-8913040. The government has rights in this material. The first author was a Visiting Scientist at the M.I.T. Artificial Intelligence Laboratory while this work was being completed. The program for generating random CSPs was written by Karl Gevecker.

References.

[Berliner 73] Berliner, H. J. "Some necessary conditions for a master chess program *Proceedings of the 3rd International Joint Conference on Artficial Intelligence*, 1973, pp. 77-85.

[Dechter and Meiri 89] Dechter, R. and I. Meiri, "Experimental evaluation of preprocessing techniques in constraint satisfaction problems," *Proceedings of the 11th International Joint Conference on Artficial Intelligence*, Vol. 1, 1989, pp. 271-277.

[Freuder 82] Freuder, E. C., "A sufficient condition for backtrack-free search," *Journal of the ACM*, Vol. 29, No. 1, 1982, pp. 24-32.

[Golomb and Baumert 65] Golomb, S. W., and L. D. Baumert, "Backtrack programming," *Journal of the ACM*, Vol. 12, No. 4, 1965, pp 516-524.

[Haralick and Elliott 80] Haralick, R. M. and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, Vol. 14, No. 3, 1980, pp. 263-313.

[Knuth 75] Knuth, D. E., "Estimating the efficiency of backtrack programs," *Mathematics of Computation*, Vol. 29, No. 129, 1975, pp. 121-136.

[Mackworth 77] Mackworth, A. K., "Consistency in networks of relations," *Artificial Intelligence*, Vol. 8, No. 1, 1977, pp. 99-118.

[Mackworth and Freuder 85] Mackworth, A. K. and E. C. Freuder, "The complexity of some polynomial network consistency algorithms for constraint satisfaction problems," *Artificial Intelligence*, Vol. 25, No. 1, 1985, pp. 65-74.

[Mohr and Henderson 86] Mohr, R. and T. C. Henderson, "Arc and path consistency revisited," *Artificial Intelligence*, Vol. 28, No. 2, 1986, pp. 225-233.

[Nadel 89] Nadel, B., "Constraint satisfaction algorithms," *Computational Intelligence*, Vol. 5, No. 4, 1989, pp. 188-224.

[Rossi, Dhar and Petrie 89] Rossi, F., V. Dhar and C. Petrie, On the Equivalence of Constraint Satisfaction Problems. MCC Technical Report ACT-AI-222-89, MCC, Austin, TX.

[Wallace and Freuder 91] Wallace, R. J. and E. C. Freuder, Heuristics for arc consistency algorithms. Technical Report No. 91-02, Department of Computer Science, University of New Hampshire, Durham, NH.